Revision: 1.2
Date: 2008/02/28 12:15:10

Contents

1 Zadania

1.1 ZADANIE 1: Piaskownica Java™ (ang. *Java™ sandbox*) (5 marca 2008)

1.1.1 Ćwiczenie 1 (1 pkt)

Napisz aplet, który odczytuje i zapisuje zawartość określonego pliku na dysku oraz wypisuje wszystkie właściwości (ang. *properties*) **Java™ Runtime Environment**. Spróbuj uruchomić ten aplet i wykorzystać jego funkcje. Dlaczego próba zakończyła się niepowodzeniem? Stwórz ręcznie lub przy pomocy narzędzia `policytool` plik polityki bezpieczeństwa pozwalający temu apletowi na wykorzystanie jego funkcjonalności.

1.1.2 Ćwiczenie 2 (1 pkt)

Napisz aplikację o funkcjonalności identycznej z apletem z poprzedniego ćwiczenia. Spróbuj uruchomić ją z domyślną polityką bezpieczeństwa. Stwórz dwie polityki bezpieczeństwa:

- jedną zabraniającą tej konkretnej aplikacji dostępu do dysku,
- drugą zabraniającą wszystkim aplikacjom oprócz niej dostępu do właściwości **JRE**.

1.1.3 Ćwiczenie 3 (1 pkt)

Zmodyfikuj aplikację z ćwiczenia 1.1.2 tak, aby przy uruchamianiu sprawdzała przyznane jej uprawnienia i informowała użytkownika w sytuacji, gdy któreś z uprawnień niezbędnych jej do działania nie zostało jej przyznane. (**PODPOWIEDŹ:** można to zrobić na kilka różnych sposobów)

1.1.4 Ćwiczenie 4 (1 pkt)

Uruchom aplikację z ćwiczenia 1.1.3 korzytając z **Java™ Web Start** tworząc w tym celu odpowiedni deskryptor **Java™ Network Launching Protocol**.

1.1.5 Przydatne źródła informacji

- Wykład z TBO
- http://www.javaworld.com/javaworld/jw-12-2000/jw-1215-security.html
- http://www.javaworld.com/channel_content/jw-security-index.shtml
- http://www.devx.com/Java/Article/27962/1954?pf=true
- http://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html
- http://en.wikipedia.org/wiki/Java_Web_Start
- http://java.sun.com/docs/books/tutorial/deployment/webstart/deploying.html
- http://java.sun.com/products/javawebstart/download-spec.html
- http://java.sun.com/developer/technicalArticles/WebServices/JWS_2/JWS_White_Paper.pdf

1.2 ZADANIE 2: Java™ Authentication and Authorization Service(12 marca 2008)

1.2.1 Ćwiczenie 1 (5 pkt)

Uzupełnij aplikację z ćwiczenia 3 o następujące elementy:

- Moduł uwierzytelnienia użytkownika, pozwalający ustalić jego tożsamość na podstawie dostępnych informacji (np. zestawu login/hasło);
- Odpowiednią klasę implementującą interfejs `java.security.Principal`;
- Odpowiedni plik konfiguracyjny dla Security Managera, wskazujący na uwierzytelnienie przy pomocy opracowanego modułu jako obowiązkową;
- Klasy, definiujące działania, które powinny wymagać odpowiednich uprawnień (dostęp do pliku itp.), implementujące interfejs `java.security.PrivilegedAction`;
- Deklaracje w pliku polityki bezpieczeństwa, pozwalające użytkownikowi na wykonanie określonych akcji tylko po uwierzytelnieniu. Deklaracje powinny wskazywać użytkownika uprawnionego do wykonywania danych operacji (ang. *principal-based policy file statements*);
- Odpowiednie modyfikacje w samej aplikacji, wykorzystujące stworzone elementy. Wynikiem powinna być aplikacja pozwalająca na wykonywanie wybranych czynności tylko pod kontrolą Security Managera poprawnie uwierzytelnionym użytkownikom.

1.2.2 Przydatne źródła informacji

- Do wykonania ćwiczeń przydatna będzie wiedza zawarta w rozdziałach Authentication i Authorization tutoriala znajdującego się pod adresem: http://java.sun.com/j2se/1.5.0/docs/guide/security/jgss/tutorials
  UWAGA: tutorial zakłada użycie systemu Kerberos do uwierzytelnienia użytkownika.

1.3 ZADANIE 3: Code-Access Security(19 marca 2008)

1.3.1 Ćwiczenie 1 (5 pkt)

Stwórz projekt w **.NET**, składający się z dwóch Assemblies:

1. Biblioteki, która udostępnia następującą funkcjonalność:
   (a) zapis i odczyt zawartości określonego pliku z dysku;
   (b) zapis i odczyt pliku z **Isolated Storage**;
   (c) odczyt zawartości jednego z kluczy rejestru systemowego, np. \\HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor \0\ProcessorNameString
2. Każda z metod powinna być zaimplementowana w dwóch wersjach:
   (a) określającej deklaratywnie (przy pomocy atrybutów) wymagane uprawnienia
   (b) określającej imperatywnie (przy pomocy metod) wymagane uprawnienia
3. Aplikacji, która wykorzystuje funkcjonalność biblioteki z punktu 1. Powinna umożliwiać wywołanie każdej z metod tej biblioteki, zarówno w wersji (1) imperatywnej, jak i (2) deklaratywnej Wywołanie powinno przebiegać wzdłuż całego łańcucha metod, które modyfikują uprawnienia tak, aby pokazać następujące przypadki:
   (a) jedna z metod w łańcuchu wywołań odbiera (DENY) konkretne uprawnienie;
   (b) jedna z metod odbiera uprawnienie, druga je bezwarunkowo przyznaje (ASSERT);
   (c) żadna z metod nie zabrania wykonania konkretnej operacji;
4. Zweryfikuj działanie projektu
5. Przy pomocy narzędzia **.NET Framework Configuration** stwórz nowy zestaw uprawnień, który zabrania Twojej aplikacji wykonania określonej operacji (wykonywanej przez bibliotekę z punktu 1). Operacja ta powinna należeć do „niezabronionych” przez łańcuch wywołań w aplikacji z punktu 3.

1.3.2 Przydatne źródła informacji

Do wykonania ćwiczeń przydatna będzie wiedza zawarta w artykułach:

- http://www.codeproject.c om/dotnet/UB_CAS_NET.asp.

1.4 ZADANIE 4: AppDomain (2 kwietnia 2008)

1.4.1 Ćwiczenie 1 (5 pkt)

Rozszerz bibliotekę z zadania ćwiczenia 5 dodając do niej kontrolkę — w zupełności wystarczy prosta etykieta (ang. *label*). Biblioteka powinna żądać następujących uprawnień:

- `RequestMinimum` do zapisu i odczytu plików na dysku;
- `RequestMinimum` do odczytu danych z rejestru;
- `RequestMinimum` do zapisu i odczytu danych z **Isolated Storage**;
- `RequestOptional` do `UIPermission`.

Następnie zmodyfikuj swoją aplikację z ćwiczenia 1.3 tak, aby po każdym wciśnięciu przycisku wywołującego którąkolwiek akcję wykonywane były następujące czynności:

- biblioteka jest ładowana do osobnego **AppDomain**;
- zestaw odpowiednich uprawnień zostaje przypisany do **AppDomain** biblioteki;
- odpowiednia operacja jest wywoływana;
- biblioteka zostaje usunięta z pamięci przez usunięcie **AppDomain**.

Co się stanie, gdy bibliotece zostanie odebrane uprawnienie `UIPermission`?

1.4.2 Przydatne źródła informacji

Do wykonania ćwiczeń przydatna będzie wiedza zawarta w artykułach:

- http://www.west-wind.com/presentations/DynamicCode/DynamicCode.htm
- http://www.gotdotnet.com/team/clr/AppdomainFAQ.aspx

1.5 ZADANIE 5: Isolated Storage (9 kwietnia 2008)

1.5.1 Ćwiczenie 1 (5 pkt)

Zbuduj dwie prostych aplikacji **Windows Forms** współdzielące jeden katalog w izolowanym magazynie danych (ang. *isolated storage*). Aby tego dokonać musisz stworzyć dwie aplikacje oraz bibliotekę, która będzie udostępniać odpowiedni zestaw operacji na na zasobach **Isolated Storage**.

Pierwsza aplikacja (edytor) powinna:

- umożliwiać stworzenie folderu o podanej nazwie we wskazanym przez użytkownika miejscu **Isolated Storage** (użytkownik powinien podawać ścieżkę relatywną do udostępnionego folderu izolowanego magazynu);
- umożliwiać stworzenie pliku o podanej nazwie i zawartości we wskazanym przez użytkownika miejscu **Isolated Storage** (użytkownik powinien podawać ścieżkę relatywną do udostępnionego folderu izolowanego magazynu).

Z kolegi druga aplikacja (przeglądarka) powinna odpowiednio:

- wyświetlać zawartość IsolatedStorage lub wybranego folderu w **Isolated Storage**;
- wyświetlać zawartość wybranego pliku z **Isolated Storage**;
- wyświetlać ilość miejsca pozostałego w **Isolated Storage**.

Biblioteka współdzielona przez obydwie wymienione aplikacje powinna:

- udostępniać odpowiedni zestaw metod wywoływanych przez edytor lub/i przeglądarkę — obydwie aplikacje powinny wykorzystywać bibliotekę w celu uzyskania dostępu do „wspólnego” izolowanego magazynu.

W celu uzyskania kompletu punktów należy:

- dostarczyć obydwie aplikacje: (1) edytor i (2) przeglądarkę, oraz współdzieloną bibliotekę
- ustalić, jakie ustawienia uprawnień do izolowanego magazynu: (1) umożliwiają aplikacjom współdzielenie danych, oraz z drugiej strony (2) uniemożliwiają tego rodzaju współdziałanie.

1.6 ZADANIE 6: Wsparcie Role-Based Security w ASP.NET 2.0 (16 kwietnia 2008)

1.6.1 Ćwiczenie 1 (7 pkt)

Stwórz aplikację **ASP.NET 2.0** wykorzystującą wbudowane mechanizmy uwierzytelnienia i autoryzacji użytkownika. Projekt powinien:

1. Wykorzystywać tryb uwierzytelnienia `Forms` (nie `Windows`) — domyślny tryb uwierzytelnienia można zmienić:
   - za pomocą narzędzia **ASP.NET Configuration** stanowiącego część **Visual Studio .NET**, lub
   - odpowiednio modyfikując zapisy w pliku `web.config` w głównym folderze projektu.

   W opisany sposób można również określić role użytkowników aplikacji.
2. Przechowywać katalog użytkowników aplikacji w bazie danych **Microsoft SQL Server 2000**. Wspomniany katalog powinien zawierać kilku zdefiniowanych użytkowników wraz z przydzielonymi im rolami;
3. Zawierać podkatalog ze stroną wyświetlającą nazwę zalogowanego użytkownika (odczytaną programowo). Dostęp do tego podkatalogu powinien być ograniczony:
   - jedna z ról występujących w aplikacji powinna mieć zabroniony dostęp do jego zawartości;
   - podobnie jeden z użytkowników, któremu przypisano role pozwalającymi na dostęp do tego podkatalogu powinien mieć odebrane do niego uprawnienia.
   - użytkownicy, którzy nie przeszli pomyślnie procedury uwierzytelnienia również nie mogą mieć dostępu do zawartości wspomnianego podkatalogu.

   Użytkownik pozbawiony praw dostępu powinien być przekierowany do strony logowania. Strona powinna też sprawdzać rolę użytkownika (przy pomocy `User.IsInRole`) — użytkownik występujący w określonej roli powinien dostać dodatkową informację tekstową.
4. Zawierać stronę startową —- powinna ona umożliwiać zalogowanie się oraz stworzenie nowego użytkownika przy pomocy wbudowanych kontrolek. Dostarczone rozwiązanie powinno zawierać zarówno kod źródłowy projektu wraz z wszystkimi niezbędnymi plikami, jak i plik bazy danych zawierający informacje niezbędne do uwierzytelnienia i autoryzacji.

1.6.2 Przydatne źródła informacji

- *Beginning **ASP.NET 2.0** in **C#** From Novice to Professional*, Apress (rozdziały 18 oraz 19)
- http://weblogs.asp.net/scottgu/archive/2006/02/24/438953.aspx

1.7 ZADANIE 7: Bardziej zaawansowane wykorzystanie Role-Based Security (23 kwietnia 2008)

1.7.1 Ćwiczenie 1 (7 pkt)

Stwórz własne rozwiązanie ograniczające dostęp użytkownika do funkcjonalności aplikacji na podstawie jego uprawnień. Rozwiązanie powinno wykorzystywać mechanizm **Role-Based Security** dostępny w **.NET Framework**.

1. Stwórz własny katalog użytkowników (w postaci pliku **XML**, bądź bazy danych zarządzanej przez **Microsoft Access** albo **Microsoft SQL Server**) przechowujący informację nt. użytkowników oraz ich przynależności do grup. Minimalny zestaw przechowywanych informacji widoczny jest na (rys. 1). Baza danych użytkowników powinna zawierać dwie grupy: (1) `users` — zwykli użytkownicy, oraz (2) `admins` — administratorzy. W bazie powinno znaleźć się 4 użytkowników:
   (a) `User1` — członek grupy `users`;
   (b) `Admin1` — członek grupy `admins`;
   (c) `Superadmin` — członek obu grup;
   (d) `Nogroup` — nie powinien należeć do żadnej z grup

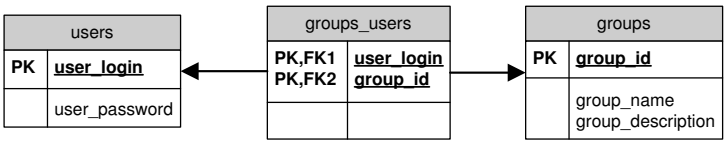

Figure 1: Przykładowy schemat bazy danych przechowującej informację o użytkownikach aplikacji

2. Stwórz formularz **Windows Forms**, który posłuży do uwierzytelnienia użytkownika — dostarczone przez użytkownika dane uwierzytelniając (ang. *credentials*) powinny być zweryfikowane i na ich podstawie powinien zostać stworzony obiekt `GenericPrincipal` reprezentujący użytkownika w ramach środowiska. Po weryfikacji tożsamości oczom użytkownika powinien ukazać się interfejs graficzny opisany poniżej.

3. Zmodyfikuj bibliotekę rozwijaną w ramach poprzednich zadań (patrz. 1.3) tak, aby ograniczała ona dostęp do funkcjonalności na podstawie informacji zawartej w obiektu `Principal`. Biblioteka powinna używać następujących regul dostępu:
   (a) Każdy zalogowany użytkownik może korzystać z **Isolated Storage**;
   (b) Członkowie grupy `users` mogą zapisywać i odczytywać pliki z dysku;
   (c) Członkowie grupy `admins` mogą odczytywać wartość z rejestru;

   Weryfikacja uprawnień powinna odbywać się na podstawie klasy `PrincipalPermission`, którą można używać zarówno w sposó imperatywny, jak i deklaratywny.

4. Stwórz aplikację korzystającą z biblioteki opisanej powyżej. Formularz uwierzytelniający, biblioteka i aplikacja powinny znajdować się w osobnych komponentach wdrożeniowych (ang. *assemblies*).

1.7.2 Przydatne źródła informacji

- http://www32.brinkster.com/srisamp/netArticles/article_13.htm
- http://www.vbip.com/books/1861007477/chapter_7477_04.asp
- http://www.codeguru.com/Csharp/.NET/net_security/authentication/article.php/c7415/
- http://www.15seconds.com/issue/041208.htm
- http://www.csharphelp.com/archives/archive195.html

1.8 ZADANIE 8: Authorization Manager (7 maja 2008)

1.8.1 Ćwiczenie 1 (12 pkt)

Stwórz aplikację **Windows Forms**, która posłuży jako przykład wykorzystania mechanizmów **Authorization Manager**. Aplikacja powinna udostępniać interfejs dla dwóch rodzajów użytkowników — zwykłego użytkownika i kierownika. Funkcjonalność udostępniona zwykłemu użytkownikowi to zgłaszanie zapotrzebowania na zakup sprzętu. Użytkownik wprowadza nazwę sprzętu oraz jego cenę. Zakup wartości do 500 EUR jest automatycznie zatwierdzany, powyżej tej kwoty wymagana jest autoryzacja przez przełożonego reprezentowanego w systemie przez rolę kierownika. Poza autoryzacją zapotrzebowania zgłoszonego przez swoich podwładnych, kierownik również może zgłaszać zapotrzebowanie na zakup (zatwierdzane automatycznie, niezależnie od kwoty).

Aplikacja powinna:

- Udostępniać odpowiedni interfejs użytkownika w zależności od uprawnień, jakie przypisane są danemu użytkownikowi (warto wykorzystać tu możliwość sprawdzenia uprawnień do wykonania danej operacji);
- Przechowywać dane o zależnościach służbowych i zapotrzebowaniu na sprzęt w bazie o schemacie przedstawionym na (rys. 2). Dla ułatwienia zakładamy płaską hierarchię służbową (tylko pracownicy i managerowie, `emp_manager` u kierownika ma wartość `null`). `approver_id` w niezatwierdzonym zamówieniu powinien mieć wartość null, zamówienia zgłoszone przez managerów powinny mieć tą samą wartość w polach `emp_id` i `approver_id` (identyfikator kierownika);
- Decyzje związane z udzielaniem (lub nie) zezwolenia na poszczególne działania powinny być zaimplementowane przy pomocy mechanizmów Authorization Managera, aplikacja powinna wywoływać odpowiednie metody AzMan.a . żadne decyzje dotyczące uprawnień (w tym weryfikacja wysokości zamówienia) nie powinny być podejmowane przez aplikację.

Optymalna kolejność działań:

1. Stworzenie aplikacji udostępniającej zadaną funkcjonalność, nie weryfikującej żadnych uprawnień (tzn. aplikacja powinna umożliwiać zapis i odczyt danych do/z bazy danych). Na tym etapie warto na formularzu pracownika umieścić jakiś element pozwalający zdecydować, czy zapotrzebowanie ma być autoryzowane automatycznie, czy pozostawione do decyzji kierownika.
2. Przetestowanie działania zbudowanego rozwiązania.
3. Właściwe skonfigurowanie **AzMan** i uzupełnienie aplikacji o kod wykorzystujący **AzMan**.

Miejsca wymagające autoryzacji przez **Authorization Manager**:

1. Decyzja o automatycznej autoryzacji (lub jej braku) zlecenia złożonego przez pracownika zrealizowana w oparciu o regułę biznesową.
2. Decyzja o automatycznej autoryzacji zlecenia złożonego przez kierownika. Możliwe rozwiązania: (1) stworzenie reguły statycznej reguły lub (2) rozszerzenie **BizRule** opisanej powyżej.
3. Zezwolenie na wykonanie operacji autoryzacji zakupu przez kierownika: (1) reguła statyczna lub (2) osobna reguła biznesowa.

Ważne: W zadaniu można wykorzystać mechanizmy autoryzacyjne **Microsoft Windows (9x/Me/NT/2000/XP/2003 Server)**, lub też własne (np. stworzone w poprzednim zadaniu). W wypadku wykorzystania mechanizmów wbudowanych w system operacyjny konieczne będzie założenie przynajmniej 2 kont użytkowników (1 dla pracownika, 1 dla kierownika). Należy pamiętać o powiązaniu nazwy użytkownika z bazą używaną w zadaniu (pole `emp_name`).

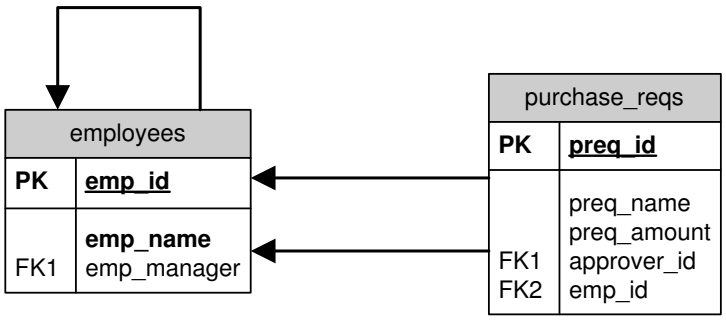

Figure 2: Schemat bazy danych rejestrującej informację o złożonych zamówieniach z uwzględnieniem danych osoby: (1) składającej, oraz (2) zatwierdzającej zapotrzebowanie

1.8.2 Przydatne źródła informacji

- http://technet2.microsoft.com/WindowsServer/en/Library/72b55950-86cc-4c7f-8fbf-3063276cd0b61033.mspx
- http://msdn.microsoft.com/msdnmag/issues/03/11/AuthorizationManager/
- http://pluralsight.com/wiki/default.aspx/Keith.GuideBook/WhatIsAuthorizationManager.html
- http://support.microsoft.com/default.aspx?scid=kb;en-us;324470
- http://dotnetjunkies.com/WebLog/davidb/archive/2005/03/26/61435.aspx

1.9 ZADANIE 9: Kryptografia w Java™ (14 maja 2008)

1.9.1 Ćwiczenie 1 (10 pkt)

Stwórz aplikację **Java™**, która będzie dostarczać funkcjonalność narzędzia `keytool` dostarczanego w ramach **Java™ Runtime Environment** rozszerzoną o możliwości: (1) generowania skrótów wiadomości, (2) generowania kodu uwierzytelnienia wiadomości, oraz (3) podpisywania danych zapisanych w pliku.

Twoja aplikacja powinna pozwalać na:

1. Stworzenie nowego klucza i dołączenie go do magazynu kluczy (ang. *keystore*). Aplikacja powinna wspierać generowanie zarówno kluczy (1) symetrycznych, jak i (2) asymetryczne, a także ochronę kluczy prywatnych i tajnych za pomocą hasła;
2. Generowanie wniosków o podpisanie certyfikatu (ang. *certification signing requests*) w formacie zgodnym ze standardem **PKCS#10** na podstawie kluczy prywatnych przechowywanych w magazynie kluczy;
3. Import podpisanych certyfikatów lub łańcuchów certyfikatów;
4. Generowanie (1) skrótu wiadomości, (2) kodu uwierzytelnienia wiadomości, oraz (3) podpisu danych przechowywanych w plikach. Osoby bardziej ambitne mogą pokusić się o zapis np. danych podpisanych w formatach przenośnych jak np. **Cryptographic Message Syntax** opisanym w standardzie **PKCS#7**.

UWAGA: Intefejs programistyczny Java 2 Platform, Standard Edition nie daje możliwośći podpisywania wniosków certyfikacji X.509, konieczne może być zatem (1) wykorzystanie zewnętrznego narzędzia, np. OpenSSL, badź (2) skorzystanie z funkcjonalności innych dostawców bezpieczeństwa, np. Bouncy Castle (http://www.bouncycastle.org/).

1.9.2 Przydatne źródła informacji

- http://java.sun.com/docs/books/tutorial/security1.2/
- http://www-106.ibm.com/developerworks/edu/j-dw-javasec1-i.html
- http://www.informit.com/articles/article.asp?p=170967&rl=1
- http://www.developer.com/java/other/article.php/631301
- http://www.developer.com/java/ent/article.php/3105261
- http://www.bouncycastle.org/
- Przykłady wykorzystania dostawcy **Bouncy Castle** zamieszczone w książce *Beginning Cryptography with **Java™*** dostępne pod adresem: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0764596330,descCd-DOWNLOAD.html

1.10 ZADANIE 10: Kryptografia w Microsoft .NET (21 maja 2008)

1.10.1 Ćwiczenie 1 (7 pkt)

Wykorzystaj wspracie dla kryptografii w interfejsie programistycznym udostępnianym przez **Microsoft .NET** do zbudowania aplikacji o funkcjonalności zbliżonej tej opisanej w treści zadania 1.9. Zasadnicze różnice w porównaniu z rozwiązaniem stworzonego w ramach zadania 1.9 będą następujące:

1. do przechowywania par składających się: (1) z klucza prywatny, oraz (2) certyfikatu **X.509** aplikacja będzie wykorzystywała systemowy magazyn kluczy **Microsoft Windows** reprezentowany przez klasę `System.Security.Cryptography.X509Certificates.X509Store`;
2. systemowy magazyn kluczy jest przeznaczony wyłącznie do przechowywania certyfikatów **X.509**, którym opcjonalnie mogą towarzyszyć odpowiadające im klucze prywatne. W miejsce funkcjonalności umożliwiającej zapis w magazynie klucza tajnego, aplikacja powinna dawać użytkownikowi możliwość:
   (a) podpisywania oraz tworzenia kopert cyfrowych na podstawie zawartości plików, oraz
   (b) utrwalania tak zabezpieczonej treści w postaci (1) dokumentów **XML** zgodnie ze standardami **XML Signature**, oraz **XML Encryption**, a także w formacie (2) tzw. danych podpisanych (ang. *signed data*), oraz kopert (ang. *enveloped data*) opisanych w standardzie **PKCS#7**.
3. aplikacja będzie pozbawiona możliwości tworzenia wniosków certyfikacji **PKCS#10** oraz ich podpisywania ze względu na brak bezpośredniego wsparcia tej funkcjonalności w interfejsie programistycznym **Microsoft .NET Framework**.

(**PODPOWIEDŹ:** Proszę zwrócić uwagę na „nieintuicyjną” nazwę klasy `X509Certificate2` reprezentującej faktycznie pozycję systemowego magazynu kluczy a nie tylko sam certyfikat **X.509** — instancja klasy `X509Certificate2` może zawierać również klucz prywatny.)

(**PODPOWIEDŹ:** Ze względu na brak możliwości tworzenia wniosków certyfikacji oraz ich podpisywania korzystne może okazać się rozszerzenie rozwiązania zadania 1.9 o możliwość eksportu pozycji klucza do pliku wymiany danych o podmiocie zgodnego z formatem **PKCS#12**. Zawartość pliku wymiany (lub inaczej magazyn) **PKCS#12** może być następnie zaimportowana do systemowego magazynu kluczy m.in. za pomocą **Microsoft Internet Explorer**. Rozszerzenie dotychczasowej funkcjonalności aplikacji stworzonej w ramach zadania 1.9 będzie wiązało się ze podniesieniem oceny za to zadanie o kolejne 2 pkt.)

1.10.2 Przydatne źródła informacji

- http://www.codeguru.pl/article-567.aspx
- http://www.codeproject.com/vb/net/PKSCStandard.asp
- http://msdn.microsoft.com/msdnmag/issues/07/03/NETSecurity/